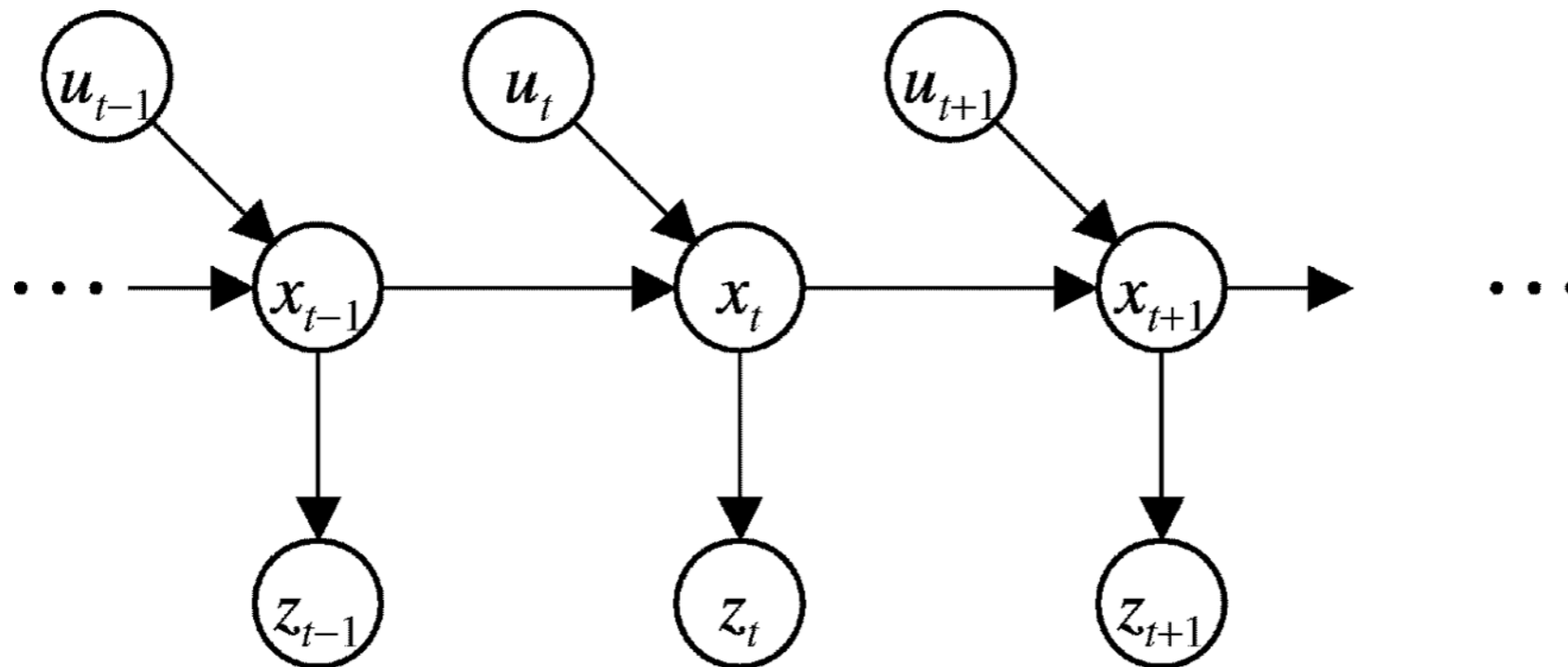




# 10. Sequential Data

# Bayes Filter (Rep.)

We can describe the overall process using a *Dynamic Bayes Network*:



- This incorporates the following Markov assumptions:

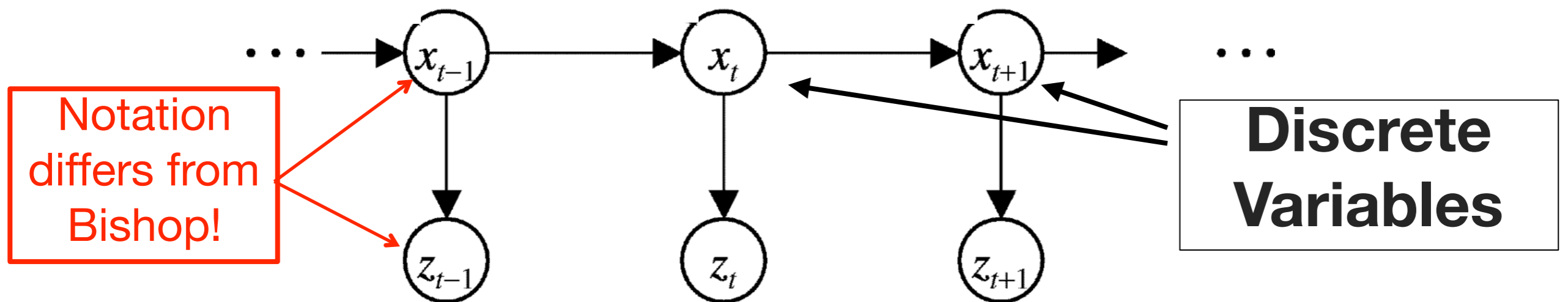
$$p(z_t \mid x_{0:t}, u_{1:t}, z_{1:t}) = p(z_t \mid x_t) \quad (\text{measurement})$$

$$p(x_t \mid x_{0:t-1}, u_{1:t}, z_{1:t}) = p(x_t \mid x_{t-1}, u_t) \quad (\text{state})$$



# Bayes Filter Without Actions

Removing the action variables we obtain:



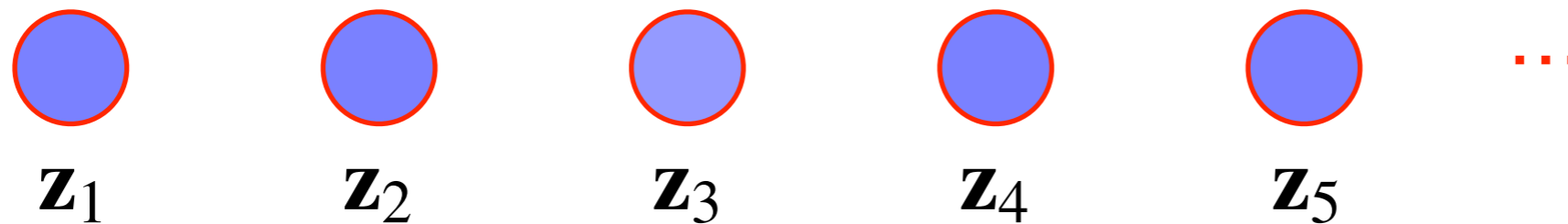
- This incorporates the following Markov assumptions:

$$p(z_t \mid x_{0:t}, z_{1:t}) = p(z_t \mid x_t) \text{ (measurement)}$$
$$p(x_t \mid x_{0:t-1}, z_{1:t}) = p(x_t \mid x_{t-1}) \text{ (state)}$$

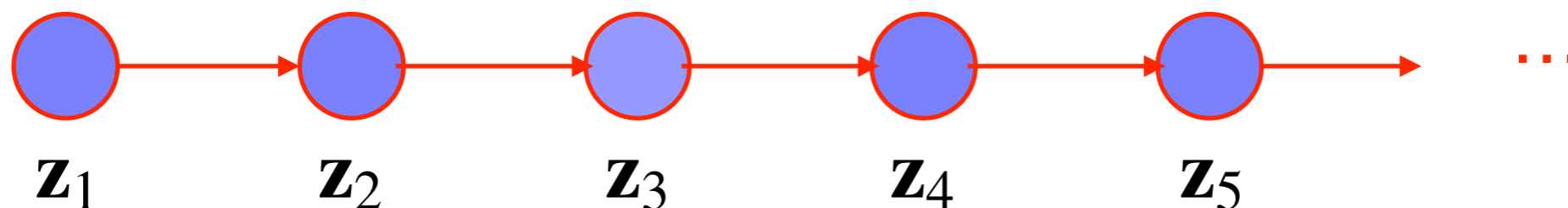


# A Model for Sequential Data

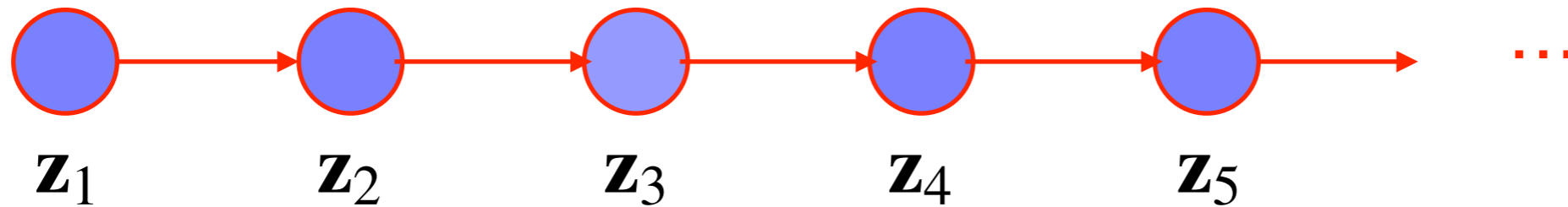
- Observations in sequential data should not be modeled as independent variables such as:



- Examples: weather forecast, speech, hand-written text, etc.
- The observation at time  $t$  depends on the observation(s) of (an) earlier time step(s):



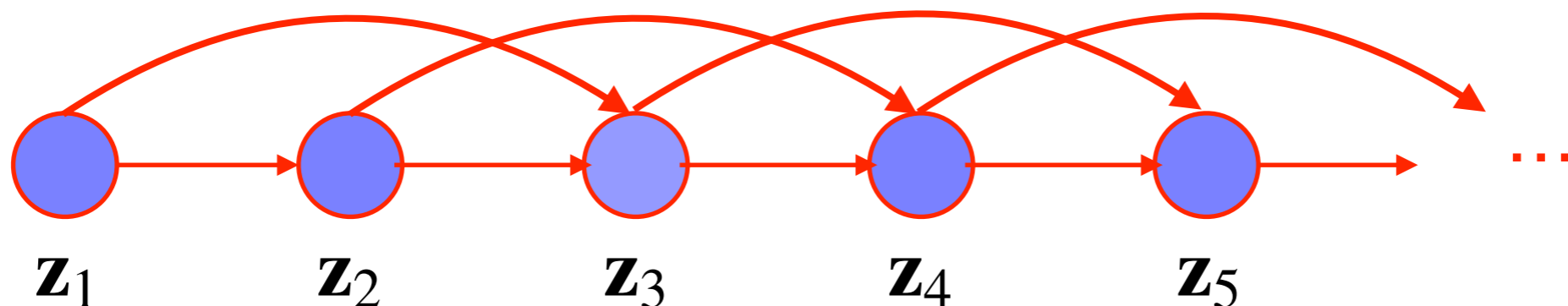
# A Model for Sequential Data



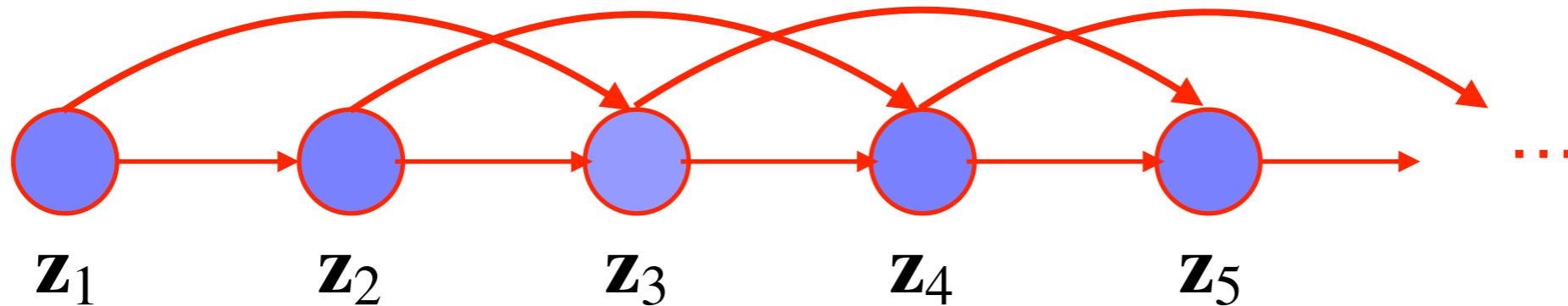
- The joint distribution is therefore (d-sep):

$$p(\mathbf{z}_1 \dots \mathbf{z}_n) = p(\mathbf{z}_1) \prod_{i=2}^n p(\mathbf{z}_i | \mathbf{z}_{i-1})$$

- **However:** often data depends on several earlier observations (not just one)



# A Model for Sequential Data



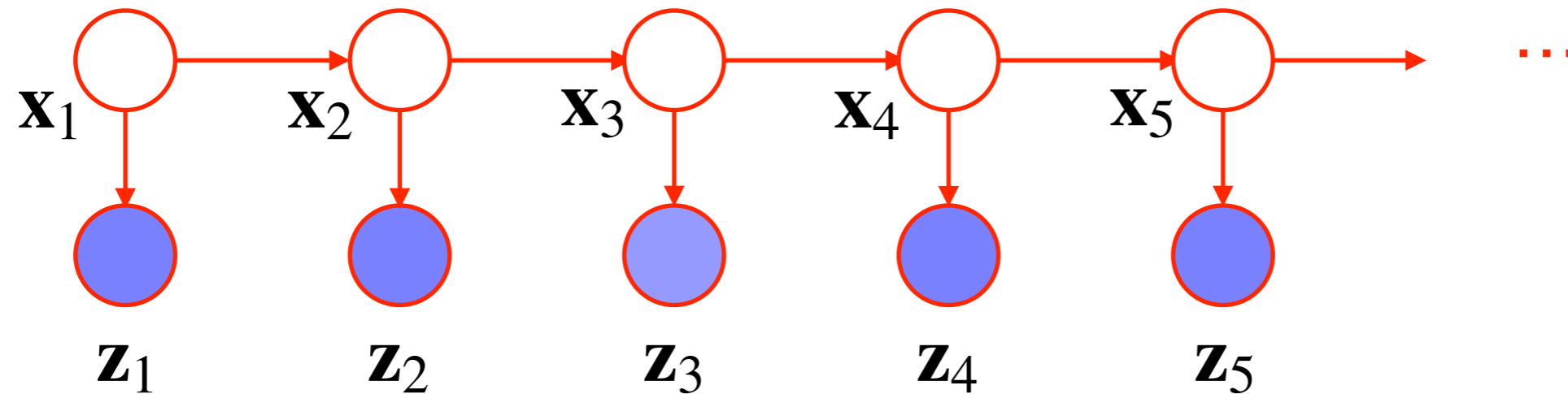
$$p(\mathbf{z}_1 \dots \mathbf{z}_n) = p(\mathbf{z}_1)p(\mathbf{z}_2 | \mathbf{z}_1) \prod_{i=3}^n p(\mathbf{z}_i | \mathbf{z}_{i-1}, \mathbf{z}_{i-2})$$

- **Problem:** number of stored parameters grows exponentially with the **order** of the Markov chain
- **Question:** can we model dependency of all previous observations with a limited number of parameters?



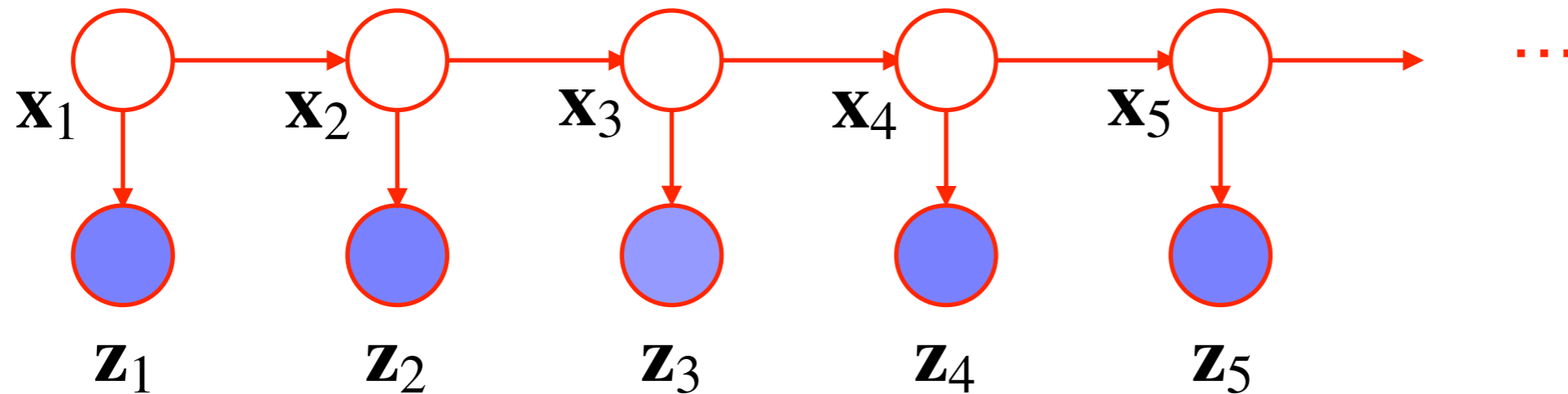
# A Model for Sequential Data

Idea: Introduce **hidden** (unobserved) variables:



# A Model for Sequential Data

Idea: Introduce **hidden** (unobserved) variables:



Now we have:  $\text{dsep}(\mathbf{x}_n, \{\mathbf{x}_1, \dots, \mathbf{x}_{n-2}\}, \mathbf{x}_{n-1})$

$$\Leftrightarrow p(\mathbf{x}_n \mid \mathbf{x}_1, \dots, \mathbf{x}_{n-2}, \mathbf{x}_{n-1}) = p(\mathbf{x}_n \mid \mathbf{x}_{n-1})$$

But:

$$\neg \text{dsep}(\mathbf{z}_n, \{\mathbf{z}_1, \dots, \mathbf{z}_{n-2}\}, \mathbf{z}_{n-1})$$

$$\Leftrightarrow p(\mathbf{z}_n \mid \mathbf{z}_1, \dots, \mathbf{z}_{n-2}, \mathbf{z}_{n-1}) \neq p(\mathbf{z}_n \mid \mathbf{z}_{n-1})$$

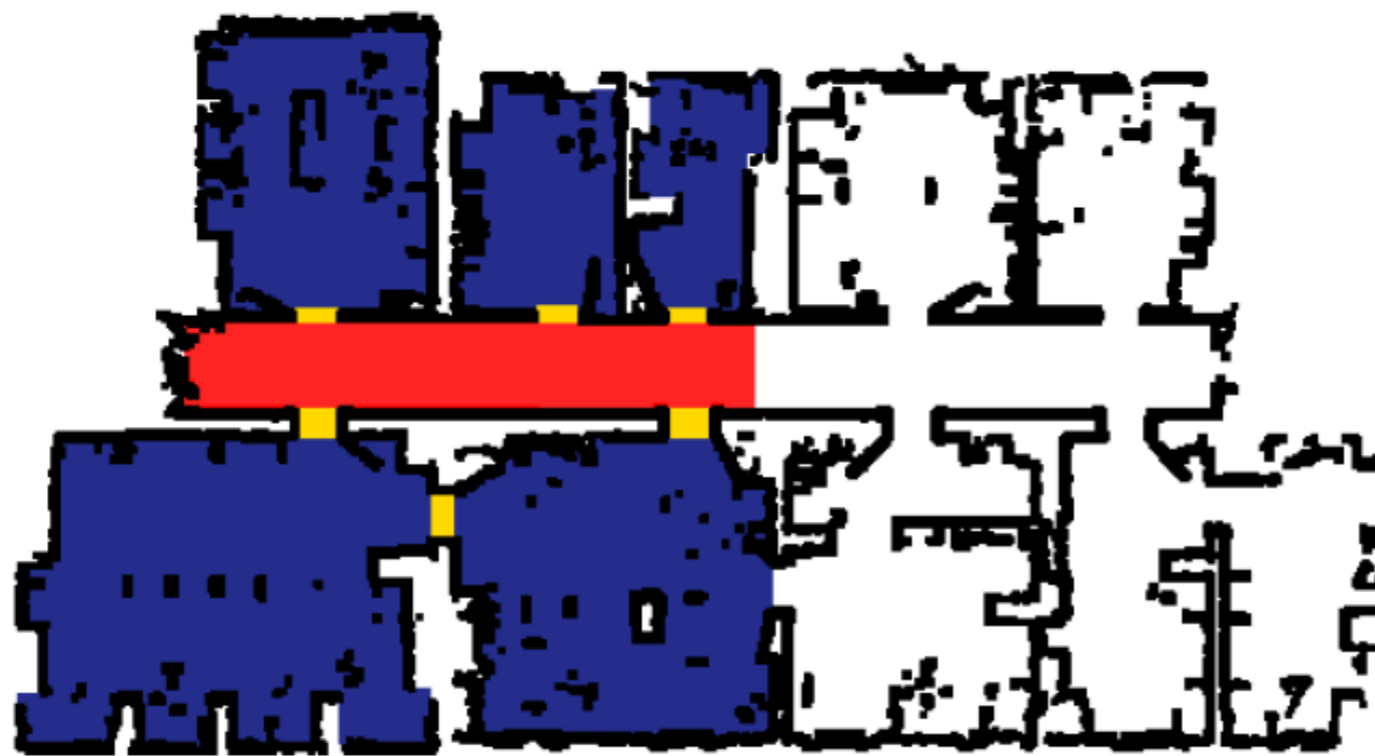
And: number of parameters is  $nK(K-1) + \text{const.}$



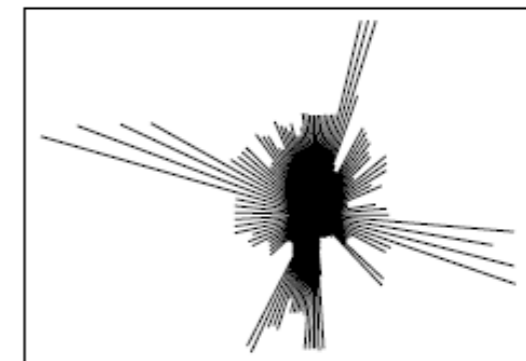
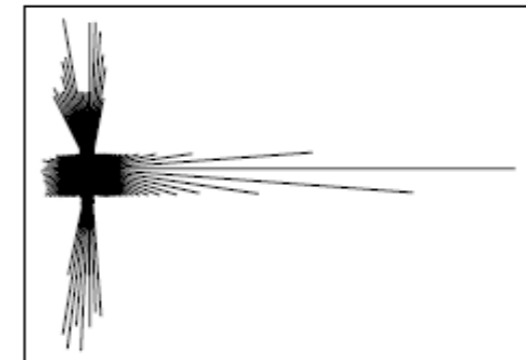


# Example

- Place recognition for mobile robots
- 3 different states: corridor, room, doorway
- Problem: misclassifications
- Idea: use information from previous time step



Corridor Room Doorway



# General Formulation of an HMM

## 1. Discrete random variables

- **Observation** variables:  $\{z_n\}$ ,  $n = 1..N$
- Discrete **state** variables (unobservable):  $\{x_n\}$ ,  $n = 1..N$
- **Number** of states  $K$ :  $x_n \in \{1..K\}$

## 2. Transition model $p(x_i | x_{i-1})$

- Markov assumption ( $x_i$  only depends on  $x_{i-1}$ )
- Represented as a  $K \times K$  **transition matrix**  $A$
- Initial probability:  $p(x_0)$  repr. as  $\pi_1, \pi_2, \pi_3$

## 3. Observation model $p(z_i | x_i)$ with parameters $\varphi$

- Observation only depends on the current state
- Example: output of a “local” place classifier

Model Parameters  
 $\theta$

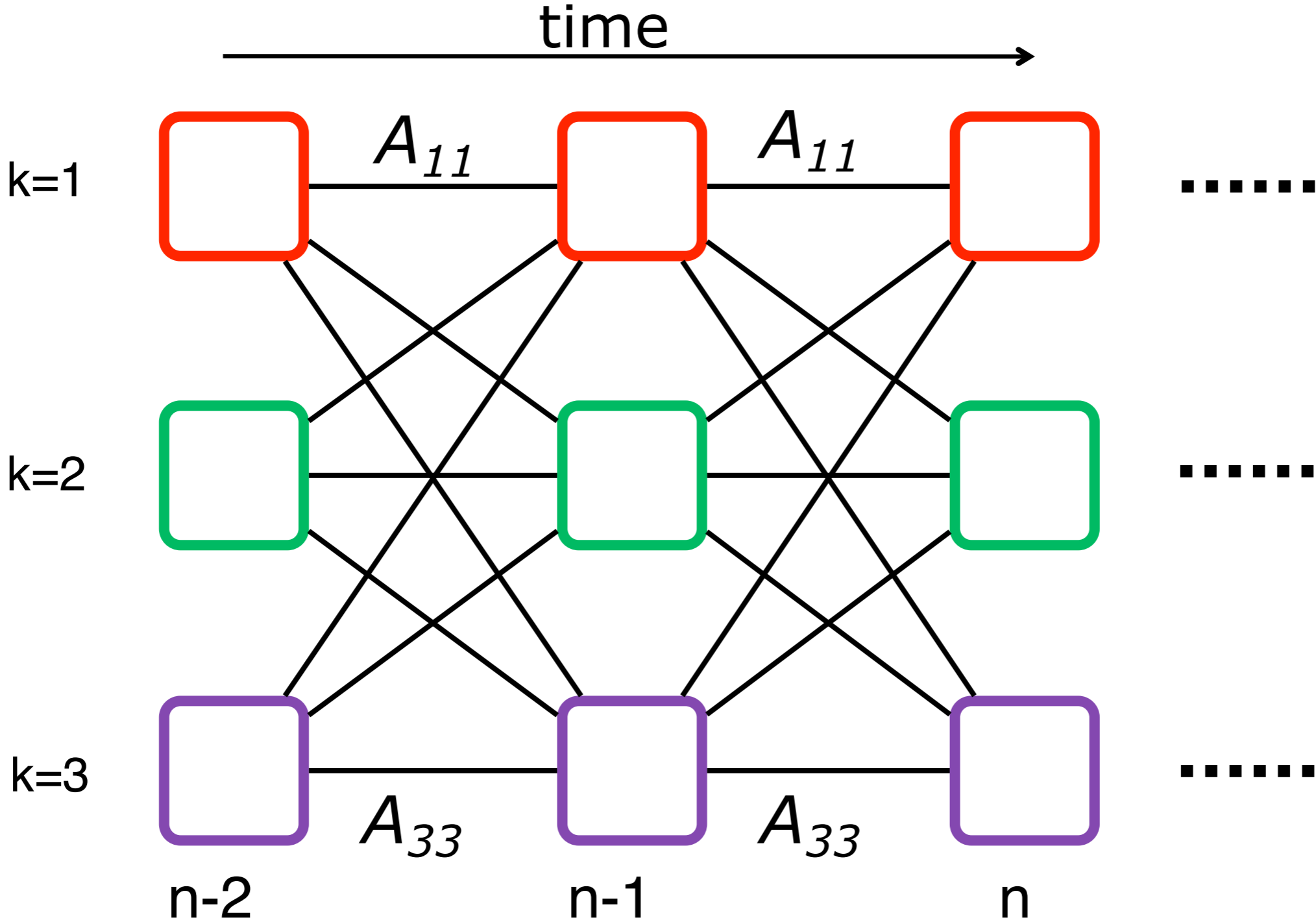
$A$

$\pi_1, \pi_2, \pi_3$

$\varphi$



# The Trellis Representation



# Application Example (1)

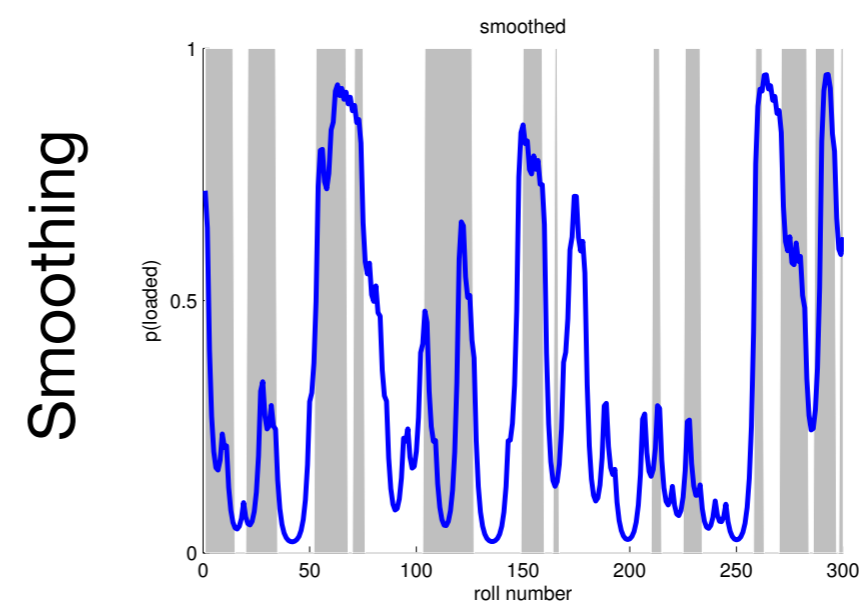
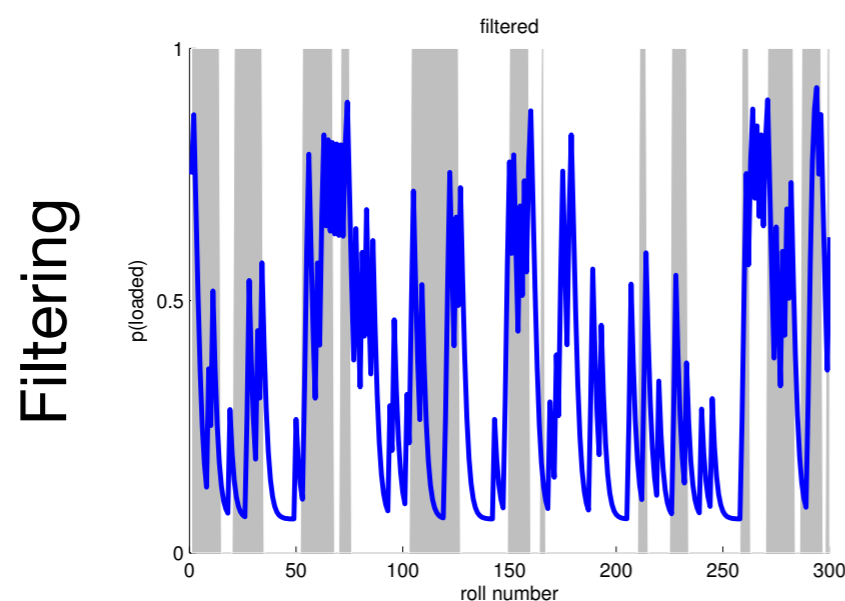
- Given an observation sequence  $\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3 \dots$
- Assume that the model parameters  $\theta = (A, \pi, \varphi)$  are known
- What is the probability that the given observation sequence is actually observed under this model, i.e. the **data likelihood**  $p(Z | \theta)$ ?
- If we are given several different models, we can choose the one with highest probability
- Expressed as a **supervised learning problem**, this can be interpreted as the inference step (classification step)



# Application Example (2)

Based on the data likelihood we can solve two different kinds of problems:

- **Filtering:** computes  $p(\mathbf{x}_n | \mathbf{z}_{1:n})$ , i.e. state probability only based on previous observations
- **Smoothing:** computes  $p(\mathbf{x}_n | \mathbf{z}_{1:N})$ , state probability based on **all** observations (including those from the future)



# Application Example (3)

- Given an observation sequence  $\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3 \dots$
- Assume that the model parameters  $\theta = (A, \pi, \varphi)$  are known
- What is the state sequence  $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3 \dots$  that **explains best** the given observation sequence?
- In the case of place recognition: which is the sequence of **truly visited places** that explains best the sequence of obtained place labels (classifications)?



# Application Example (4)

- Given an observation sequence  $\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3 \dots$
- What are the optimal model parameters  $\theta = (\Lambda, \pi, \varphi)$ ?
- This can be interpreted as the **training step**
- It is in general the most difficult problem



# Summary: 4 Operations on HMMs

1. Compute data likelihood  $p(Z|\theta)$  from a known model
  - Can be computed with the **forward** algorithm
2. Filtering or Smoothing of the state probability
  - Filtering: **forward** algorithm
  - Smoothing: **forward-backward** algorithm
3. Compute optimal state sequence with a known model
  - Can be computed with the **Viterbi**-Algorithm
4. Learn model parameters for an observation sequence
  - Can be computed using **Expectation-Maximization** (or Baum-Welch)





# The Forward Algorithm

Goal: compute  $p(Z|\theta)$  (we drop  $\theta$  in the following)

$$p(\mathbf{z}_1, \dots, \mathbf{z}_n) = \sum_{\mathbf{x}_n} p(\mathbf{z}_1, \dots, \mathbf{z}_n, \mathbf{x}_n) =: \sum_{\mathbf{x}_n} \alpha(\mathbf{x}_n)$$



# The Forward Algorithm

Goal: compute  $p(Z|\theta)$  (we drop  $\theta$  in the following)

$$p(\mathbf{z}_1, \dots, \mathbf{z}_n) = \sum_{\mathbf{x}_n} p(\mathbf{z}_1, \dots, \mathbf{z}_n, \mathbf{x}_n) =: \sum_{\mathbf{x}_n} \alpha(\mathbf{x}_n)$$

We can calculate  $\alpha$  recursively:

$$\alpha(\mathbf{x}_n) = p(\mathbf{z}_n | \mathbf{x}_n) \sum_{\mathbf{x}_{n-1}} \alpha(\mathbf{x}_{n-1}) p(\mathbf{x}_n | \mathbf{x}_{n-1})$$



# The Forward Algorithm

Goal: compute  $p(Z|\theta)$  (we drop  $\theta$  in the following)

$$p(\mathbf{z}_1, \dots, \mathbf{z}_n) = \sum_{\mathbf{x}_n} p(\mathbf{z}_1, \dots, \mathbf{z}_n, \mathbf{x}_n) =: \sum_{\mathbf{x}_n} \alpha(\mathbf{x}_n)$$

We can calculate  $\alpha$  recursively:

$$\alpha(\mathbf{x}_n) = p(\mathbf{z}_n | \mathbf{x}_n) \sum_{\mathbf{x}_{n-1}} \alpha(\mathbf{x}_{n-1}) p(\mathbf{x}_n | \mathbf{x}_{n-1})$$

This is the same recursive formula as we had in the first lecture!



# The Forward Algorithm

Goal: compute  $p(Z|\theta)$  (we drop  $\theta$  in the following)

$$p(\mathbf{z}_1, \dots, \mathbf{z}_n) = \sum_{\mathbf{x}_n} p(\mathbf{z}_1, \dots, \mathbf{z}_n, \mathbf{x}_n) =: \sum_{\mathbf{x}_n} \alpha(\mathbf{x}_n)$$

We can calculate  $\alpha$  recursively:

$$\alpha(\mathbf{x}_n) = p(\mathbf{z}_n | \mathbf{x}_n) \sum_{\mathbf{x}_{n-1}} \alpha(\mathbf{x}_{n-1}) p(\mathbf{x}_n | \mathbf{x}_{n-1})$$

This is the same recursive formula as we had in the first lecture!

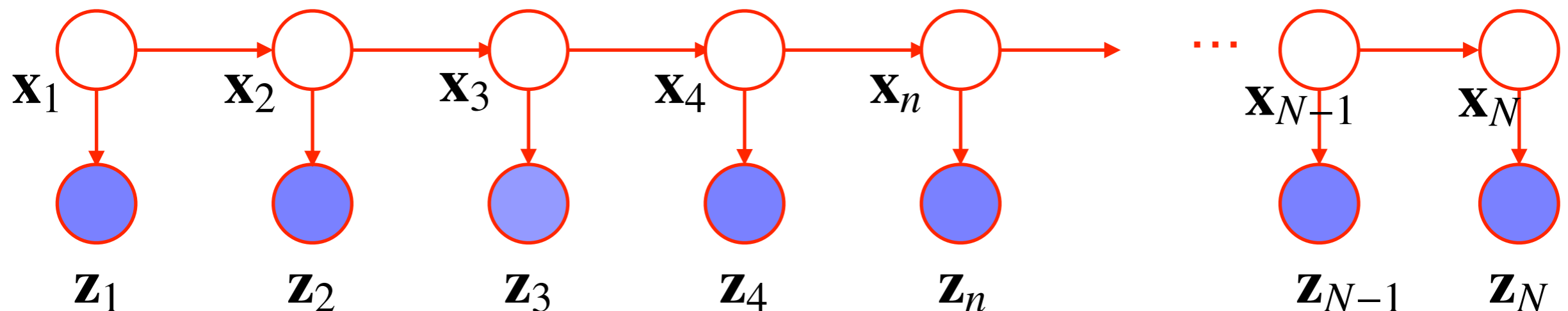
Filtering: 
$$p(\mathbf{x}_n | \mathbf{z}_1, \dots, \mathbf{z}_n) = \frac{p(\mathbf{z}_1, \dots, \mathbf{z}_n, \mathbf{x}_n)}{p(\mathbf{z}_1, \dots, \mathbf{z}_n)} = \frac{\alpha(\mathbf{x}_n)}{\sum_{\mathbf{x}_n} \alpha(\mathbf{x}_n)}$$



# The Forward-Backward Algorithm

- As before we set  $\alpha(\mathbf{x}_n) = p(\mathbf{z}_1, \dots, \mathbf{z}_n, \mathbf{x}_n)$
- We also define  $\beta(\mathbf{x}_n) = p(\mathbf{z}_{n+1}, \dots, \mathbf{z}_N | \mathbf{x}_n)$

e.g.  $n = 5$ :



# The Forward-Backward Algorithm

- As before we set  $\alpha(\mathbf{x}_n) = p(\mathbf{z}_1, \dots, \mathbf{z}_n, \mathbf{x}_n)$
- We also define  $\beta(\mathbf{x}_n) = p(\mathbf{z}_{n+1}, \dots, \mathbf{z}_N | \mathbf{x}_n)$
- This can be recursively computed (backwards):

$$\begin{aligned}\beta(\mathbf{x}_{n-1}) &= p(\mathbf{z}_n, \dots, \mathbf{z}_N | \mathbf{x}_{n-1}) \\ &= \sum_{\mathbf{x}_n} p(\mathbf{x}_n, \mathbf{z}_n, \dots, \mathbf{z}_N | \mathbf{x}_{n-1}) \\ &= \sum_{\mathbf{x}_n} p(\mathbf{z}_{n+1}, \dots, \mathbf{z}_N | \mathbf{x}_n, \cancel{\mathbf{z}_n}, \cancel{\mathbf{x}_{n-1}}) p(\mathbf{x}_n, \mathbf{z}_n | \mathbf{x}_{n-1}) \\ &= \sum_{\mathbf{x}_n} p(\mathbf{z}_{n+1}, \dots, \mathbf{z}_N | \mathbf{x}_n) p(\mathbf{z}_n | \cancel{\mathbf{x}_{n-1}}, \mathbf{x}_n) p(\mathbf{x}_n | \mathbf{x}_{n-1}) \\ &= \sum_{\mathbf{x}_n} \beta(\mathbf{x}_n) p(\mathbf{z}_n | \mathbf{x}_n) p(\mathbf{x}_n | \mathbf{x}_{n-1})\end{aligned}$$



# The Forward-Backward Algorithm

- As before we set  $\alpha(\mathbf{x}_n) = p(\mathbf{z}_1, \dots, \mathbf{z}_n, \mathbf{x}_n)$
- We also define  $\beta(\mathbf{x}_n) = p(\mathbf{z}_{n+1}, \dots, \mathbf{z}_N | \mathbf{x}_n)$
- This can be recursively computed (backwards):

$$\beta(\mathbf{x}_n) = \sum_{\mathbf{x}_{n+1}} \beta(\mathbf{x}_{n+1}) p(\mathbf{z}_{n+1} | \mathbf{x}_{n+1}) p(\mathbf{x}_{n+1} | \mathbf{x}_n)$$

- This is exactly the same as the message-passing algorithm (“sum-product”)!
  - forward messages  $\alpha_n$  (vector of length  $K$ )
  - backward messages  $\beta_n$  (vector of length  $K$ )



# Smoothing with Forward-Backward

First we compute  $p(\mathbf{x}_n, \mathbf{z}_1, \dots, \mathbf{z}_N)$ :

$$\begin{aligned} p(\mathbf{x}_n, \mathbf{z}_1, \dots, \mathbf{z}_N) &= p(\mathbf{z}_1, \dots, \mathbf{z}_N \mid \mathbf{x}_n) p(\mathbf{x}_n) \\ &= p(\mathbf{z}_1, \dots, \mathbf{z}_n \mid \mathbf{x}_n) p(\mathbf{z}_{n+1}, \dots, \mathbf{z}_N \mid \mathbf{x}_n) p(\mathbf{x}_n) \\ &= p(\mathbf{z}_1, \dots, \mathbf{z}_n, \mathbf{x}_n) p(\mathbf{z}_{n+1}, \dots, \mathbf{z}_N \mid \mathbf{x}_n) \\ &= \alpha(\mathbf{x}_n) \beta(\mathbf{x}_n) \end{aligned}$$





# Smoothing with Forward-Backward

First we compute  $p(\mathbf{x}_n, \mathbf{z}_1, \dots, \mathbf{z}_N)$ :

$$p(\mathbf{x}_n, \mathbf{z}_1, \dots, \mathbf{z}_N) = \alpha(\mathbf{x}_n)\beta(\mathbf{x}_n)$$

with that we can compute  $p(\mathbf{z}_1, \dots, \mathbf{z}_N)$ :

$$p(\mathbf{z}_1, \dots, \mathbf{z}_N) = \sum_{\mathbf{x}_n} p(\mathbf{x}_n, \mathbf{z}_1, \dots, \mathbf{z}_N) = \sum_{\mathbf{x}_n} \alpha(\mathbf{x}_n)\beta(\mathbf{x}_n)$$



# Smoothing with Forward-Backward

First we compute  $p(\mathbf{x}_n, \mathbf{z}_1, \dots, \mathbf{z}_N)$ :

$$p(\mathbf{x}_n, \mathbf{z}_1, \dots, \mathbf{z}_N) = \alpha(\mathbf{x}_n)\beta(\mathbf{x}_n)$$

with that we can compute  $p(\mathbf{z}_1, \dots, \mathbf{z}_N)$ :

$$p(\mathbf{z}_1, \dots, \mathbf{z}_N) = \sum_{\mathbf{x}_n} p(\mathbf{x}_n, \mathbf{z}_1, \dots, \mathbf{z}_N) = \sum_{\mathbf{x}_n} \alpha(\mathbf{x}_n)\beta(\mathbf{x}_n)$$

and finally:

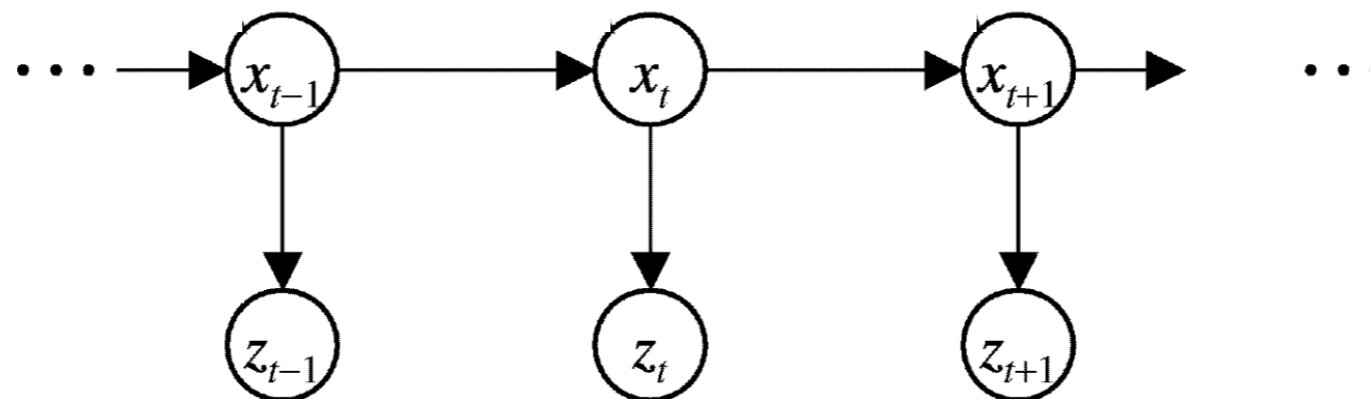
$$p(\mathbf{x}_n \mid \mathbf{z}_1, \dots, \mathbf{z}_N) = \frac{p(\mathbf{x}_n, \mathbf{z}_1, \dots, \mathbf{z}_N)}{p(\mathbf{z}_1, \dots, \mathbf{z}_N)} = \frac{\alpha(\mathbf{x}_n)\beta(\mathbf{x}_n)}{\sum_{\mathbf{x}_n} \alpha(\mathbf{x}_n)\beta(\mathbf{x}_n)}$$



## 2. Computing the Most Likely States

- Goal: find a state sequence  $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3 \dots$  that maximizes the probability  $p(\mathbf{X}, \mathbf{Z} | \theta)$
- Define  $\delta(\mathbf{x}_n) = \max_{\mathbf{x}_1, \dots, \mathbf{x}_{n-1}} p(\mathbf{x}_1, \dots, \mathbf{x}_n | \mathbf{z}_1, \dots, \mathbf{z}_n)$

This is the probability of state  $j$  by taking the most probable path.



## 2. Computing the Most Likely States

- Goal: find a state sequence  $x_1, x_2, x_3 \dots$  that maximizes the probability  $p(\mathbf{X}, \mathbf{Z} | \theta)$
- Define 
$$\delta(\mathbf{x}_n) = \max_{\mathbf{x}_1, \dots, \mathbf{x}_{n-1}} p(\mathbf{x}_1, \dots, \mathbf{x}_n | \mathbf{z}_1, \dots, \mathbf{z}_n)$$

This can be computed recursively:

$$\delta(\mathbf{x}_n) = \max_{\mathbf{x}_{n-1}} \delta(\mathbf{x}_{n-1}) p(\mathbf{x}_n | \mathbf{x}_{n-1}) p(\mathbf{z}_n, | \mathbf{x}_n)$$

we also have to compute the argmax:

$$\psi(\mathbf{x}_n) = \arg \max_{\mathbf{x}_{n-1}} \delta(\mathbf{x}_{n-1}) p(\mathbf{x}_n | \mathbf{x}_{n-1}) p(\mathbf{z}_n, | \mathbf{x}_n)$$



# The Viterbi algorithm

- Initialize:
  - $\delta(\mathbf{x}_0) = p(\mathbf{x}_0) p(\mathbf{z}_0 | \mathbf{x}_0)$
  - $\psi(\mathbf{x}_0) = 0$
- Compute recursively for  $n=1 \dots N$ :
  - $\delta(\mathbf{x}_n) = p(\mathbf{z}_n | \mathbf{x}_n) \max_{x_{n-1}} [\delta(\mathbf{x}_{n-1}) p(\mathbf{x}_n | \mathbf{x}_{n-1})]$
  - $\psi(\mathbf{x}_n) = \operatorname{argmax}_{x_{n-1}} [\delta(\mathbf{x}_{n-1}) p(\mathbf{x}_n | \mathbf{x}_{n-1})]$
- On termination:
  - $p(\mathbf{Z}, \mathbf{X} | \theta) = \max_{x_N} \delta(x_N)$
  - $\mathbf{x}_N^* = \operatorname{argmax}_{x_N} \delta(x_N)$
- Backtracking:
  - $\mathbf{x}_n^* = \psi(x_{n+1}^*)$



# 3. Learning the Model Parameters

- Given an observation sequence  $\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3 \dots$
- Find optimal model parameters  $\theta = \pi, A, \varphi$
- We need to maximize the likelihood  $p(Z|\theta)$
- Can not be solved in closed form
- Iterative algorithm:  
Expectation Maximization (EM) or for the case of HMMs: Baum-Welch algorithm



# 3. Learning the Model Parameters

- Idea: instead of maximizing

$$p(\mathbf{z}_1, \dots, \mathbf{z}_N | \theta) = \sum_X p(\mathbf{z}_1, \dots, \mathbf{z}_N, \mathbf{x}_1, \dots, \mathbf{x}_N | \theta)$$

- we maximize the **expected** log likelihood:

$$\sum_X p(\mathbf{x}_1, \dots, \mathbf{x}_N | \mathbf{z}_1, \dots, \mathbf{z}_N, \theta) \log p(\mathbf{z}_1, \dots, \mathbf{z}_N, \mathbf{x}_1, \dots, \mathbf{x}_N | \theta)$$

- it can be shown that this is a lower bound of the actual log-likelihood  $p(Z|\theta)$
- this is exactly the Expectation-Maximization (EM) algorithm



# The Baum-Welsh algorithm

- E-Step (assuming we know  $\pi, A, \phi$ , i.e.  $\theta^{\text{old}}$ )
- Define the posterior probability of being in state  $i$  at step  $k$ :
- Define  $\gamma(\mathbf{x}_n) = p(\mathbf{x}_n | Z)$





# The Baum-Welsh algorithm

- E-Step (assuming we know  $\pi, A, \phi$ , i.e.  $\theta^{\text{old}}$ )
- Define the posterior probability of being in state  $i$  at step  $k$ :
- Define  $\gamma(\mathbf{x}_n) = p(\mathbf{x}_n | \mathbf{Z}_1, \dots, \mathbf{Z}_n)$
- It follows that  $\gamma(\mathbf{x}_n) = \alpha(\mathbf{x}_n) \beta(\mathbf{x}_n) / p(\mathbf{Z})$



# The Baum-Welsh algorithm

- E-Step (assuming we know  $\pi, A, \phi$ , i.e.  $\theta^{\text{old}}$ )
- Define the posterior probability of being in state  $i$  at step  $k$ :
  - Define  $\gamma(\mathbf{x}_n) = p(\mathbf{x}_n | \mathbf{Z}_1, \dots, \mathbf{Z}_n)$
  - It follows that  $\gamma(\mathbf{x}_n) = \alpha(\mathbf{x}_n) \beta(\mathbf{x}_n) / p(\mathbf{Z})$
  - Define  $\xi(\mathbf{x}_{n-1}, \mathbf{x}_n) = p(\mathbf{x}_{n-1}, \mathbf{x}_n | \mathbf{Z})$
  - It follows that
$$\xi(\mathbf{x}_{n-1}, \mathbf{x}_n) = \alpha(\mathbf{x}_{n-1}) p(\mathbf{z}_n | \mathbf{x}_n) p(\mathbf{x}_n | \mathbf{x}_{n-1}) \beta(\mathbf{x}_n) / p(\mathbf{Z})$$



# The Baum-Welsh algorithm

- E-Step (assuming we know  $\pi, A, \phi$ , i.e.  $\theta^{\text{old}}$ )
- Define the posterior probability of being in state  $i$  at step  $k$ :

- Define  $\gamma(\mathbf{x}_n) = p(\mathbf{x}_n | \mathbf{z}_1, \dots, \mathbf{z}_n)$

- It follows that  $\gamma(\mathbf{x}_n) = \alpha(\mathbf{x}_n) \beta(\mathbf{x}_n) / p(\mathbf{Z})$

- Define  $\xi(x_{n-1}, x_n) = p(x_{n-1}, x_n | \mathbf{Z})$

- It follows that

$$\xi(\mathbf{x}_{n-1}, \mathbf{x}_n) = \alpha(\mathbf{x}_{n-1}) p(\mathbf{z}_n | \mathbf{x}_n) p(\mathbf{x}_n | \mathbf{x}_{n-1}) \beta(\mathbf{x}_n) / p(\mathbf{Z})$$

- The algorithm maximizes:

$$Q(\theta, \theta^{\text{old}}) = \sum p(X | \mathbf{Z}, \theta^{\text{old}}) \log p(\mathbf{Z}, X | \theta)$$

**“Expected  
complete data  
log-likelihood”**



# The Baum-Welsh algorithm

- Maximizing  $Q$  also maximizes the likelihood:

$$p(Z|\theta) \geq p(Z|\theta^{\text{old}})$$

- M-Step:

$$\pi_k = \frac{\sum_{\mathbf{x}} \gamma(\mathbf{x}) x_{1k}}{\sum_{j=1} \sum_{\mathbf{x}} \gamma(\mathbf{x}) x_{1j}}$$

**here, we need forward and backward step!**

$$A_{jk} = \frac{\sum_{t=2}^T \xi(x_{t-1,j}, x_{tk})}{\sum_{l=1}^K \sum_{t=2}^T \xi(x_{t-1,j}, x_{tl})}$$

- With these new values,  $Q$  is recomputed
- This is done until the likelihood does not increase anymore (convergence)




# The Baum-Welsh Algorithm - Summary

- Start with an initial estimate of  $\theta=(\pi,A,\varphi)$   
e.g. uniformly and k-means for  $\varphi$
- Compute  $Q(\theta,\theta^{\text{old}})$  (E-Step)
- Maximize  $Q$  (M-step)
- Iterate E and M until convergence
- In each iteration one full application of the forward-backward algorithm is performed
- Result gives a **local** optimum
- For other local optima, the algorithm needs to be started again with new initialization



# The Scaling problem

- Probability of sequences

$$\prod_i p(x_i | \dots) \ll 1$$


- Probabilities are very small
- The product of the terms soon is very small
- Usually: converting to log-space works
- But: we have sums of products!
- Solution: Rescale/Normalize the probability during the computation, e.g.:

$$\hat{\alpha}(\mathbf{x}_n) = \alpha(\mathbf{x}_n) / p(\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_n)$$



# Summary

- HMMs are a way to model sequential data
- They assume discrete states
- Three possible operations can be performed with HMMs:
  - Data likelihood, given a model and an observation
  - Most likely state sequence, given a model and an observation
  - Optimal Model parameters, given an observation
- Appropriate scaling solves numerical problems
- HMMs are widely used, e.g. in speech recognition

